

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**

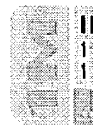
L Number	Hits	Search Text	DB	Time stamp
-	2	5889996.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 15:47
-	2	5768593.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 15:48
-	33	5768593.URPN.	USPAT	2004/07/29 15:47
-	12	5889996.URPN.	USPAT	2004/07/29 15:48
-	2	5872978.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 15:51
-	4	5872978.URPN.	USPAT	2004/07/29 15:50
-	2	4638423.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 15:52
-	33	4638423.URPN.	USPAT	2004/07/29 15:51
-	2	5872978.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 15:54
-	4	5872978.URPN.	USPAT	2004/07/29 15:52
-	13	(5768593.URPN. or 5889996.URPN. or 4638423.URPN.) and (vm or jvm or (virtual adj machine)) and interpret\$3 and native and (loop\$3 or repeat\$3) and (memory or address\$3 or range)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:28
-	11216	(wat or (way near3 ahead) or awat or (away near3 ahead) or (ahead near3 time) )	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:29
-	415	(wat or (way near3 ahead) or awat or (away near3 ahead) or (ahead near3 time) ) and (java or bytecode or byte-code) and (compil\$5 or optimiz\$5 or optimis\$5)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:30
-	35	(wat or (way near3 ahead) or awat or (away near3 ahead) or (ahead near3 time) ) and (java or bytecode or byte-code) and (compil\$5 or optimiz\$5 or optimis\$5) and 717/???ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:34
-	264	(wat or (way near3 ahead) or awat or (away near3 ahead) or (ahead near3 time) ) and (java or bytecode or byte-code) and (compil\$5 or optimiz\$5 or optimis\$5) and loop\$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:36
-	6986480	way ahead of time	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:35

-	23527	(way ahead of time) and (java or bytecode or byte-code)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:35
-	264	(wat or (way near3 ahead) or awat or (away near3 ahead) or (ahead near3 time) ) and (java or bytecode or byte-code) and (compil\$5 or optimiz\$5 or optimis\$5) and loop\$3 and (way ahead of time)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:36
-	315891	(way ahead of time).ti.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:36
-	0	((way adj ahead) near3 time).ti.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:37
-	4	((way adj ahead)).ti.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:37
-	1	((away adj ahead)).ti.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:39
-	0	toba and comil\$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:41
-	0	aot and comil\$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:41
-	28	aot and compil\$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:41
-	20	toba and compil\$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:46
-	320	717/140.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:46
-	166	717/148.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:47
-	130	717/160.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/29 16:47

-	8	(compil\$5 near3 native) same loop\$3 and (jvm or vm or bytecode or byte-code)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/08/05 13:50
-	1	5768593.pn. and loop\$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/08/05 13:46
-	1	5768593.pn. and loop\$3 and execut\$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/08/05 13:47
-	0	(compil\$5 near3 native) same loop\$3 and (backward near2 loop\$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/08/05 13:51
-	2	(compil\$5 near3 native) same loop\$3 and (backward near2 branch\$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/08/05 13:52
-	2	(compil\$5 near3 native) same loop\$3 and (conditional near2 branch\$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/08/05 13:53



Welcome  
United States Patent and Trademark Office



» Se.

[Help](#) [FAQ](#) [Terms](#) [IEEE Peer Review](#)

## Quick Links

Welcome to IEEE Xplore®

- Home
- What Can I Access?
- Log-out

## Tables of Contents

- ☐ Journals & Magazines
- ☐ Conference Proceedings
- ☐ Standards


## Search

- ☐ By Author  
☐ Basic  
☐ Advanced

## Member Services

- Join IEEE
- Establish IEEE Web Account
- Access the IEEE Member Digital Library

# IEEE Enterprise

-  Access the  
IEEE Enterprise  
File Cabinet

 **Print Format**

Your search matched **3** of **1058483** documents.

A maximum of **500** results are displayed, **50** to a page, sorted by **Relevance Descending** order.

### Refine This Search:

You may refine your search by editing the current search expression or entering a new one in the text box.

```
hybrid<and>compiler<and>loop
```

Search

☐ Check to search within this result set

### Results Key:

**JNL** = Journal or Magazine    **CNF** = Conference    **STD** = Standard

## 1 NAPA C: compiling for a hybrid RISC/FPGA architecture

*Gokhale, M.B.; Stone, J.M.;*

FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on , 15-17 April 1998

Pages:126 - 135

[\[Abstract\]](#)   [\[PDF Full-Text \(204 KB\)\]](#)   **IEEE CNF**

## 2 An object oriented dynamic simulation architecture for rapid space prototyping

Strunce, R.R., Jr.; Maher, F.H.;

Aerospace Conference Proceedings, 2000 IEEE , Volume: 11 , 18-25 March 2000  
Pages:529 - 537 vol.1

[\[Abstract\]](#)   [\[PDF Full-Text \(1372 KB\)\]](#)   **IEEE CNF**

### 3 Software pipelining for Jetpipeline architecture

Katahira, M.; Sasaki, T.; Hong Shen; Kobayashi, H.; Nakamura, T.;

Parallel Architectures, Algorithms and Networks, 1994. (ISPAN) International Symposium on , 14-16 Dec. 1994

Pages:127 - 134

[\[Abstract\]](#)   [\[PDF Full-Text \(392 KB\)\]](#)   **IEEE CNF**

IEEE HOME | SEARCH IEEE | SHOP | WEB ACCOUNT | CONTACT IEEE


[Membership](#) | [Publications/Services](#) | [Standards](#) | [Conferences](#) | [Careers/Jobs](#)

 Welcome  
 United States Patent and Trademark Office

[Help](#) | [FAQ](#) | [Terms](#) | [IEEE Peer Review](#)
[Quick Links](#)

» Search

## Welcome to IEEE Xplore®

- ☐ Home
- ☐ What Can I Access?
- ☐ Log-out

## Tables of Contents

- ☐ Journals & Magazines
- ☐ Conference Proceedings
- ☐ Standards

## Search

- ☐ By Author
- ☐ Basic
- ☐ Advanced

## Member Services

- ☐ Join IEEE
- ☐ Establish IEEE Web Account
- ☐ Access the IEEE Member Digital Library

## IEEE Enterprise

- ☐ Access the IEEE Enterprise File Cabinet

 Your search matched **2** of **1058483** documents.

 A maximum of **500** results are displayed, **50** to a page, sorted by **Relevance Descending** order.

## Refine This Search:

You may refine your search by editing the current search expression or enter a new one in the text box.


☐ Check to search within this result set

## Results Key:

**JNL** = Journal or Magazine    **CNF** = Conference    **STD** = Standard

**1 A study of the cache and branch performance issues with running Java on current hardware platforms**
*Hsieh, C.-H.A.; Conte, M.T.; Johnson, T.L.; Gyllenhaal, J.C.; Hwu, W.-M.W.; Compcon '97. Proceedings, IEEE , 23-26 Feb. 1997*  
 Pages:211 - 216

[\[Abstract\]](#)    [\[PDF Full-Text \(564 KB\)\]](#)    **IEEE CNF**
**2 Quick and easy interactive molecular dynamics using Java3D**
*Vormoor, O.;*  
 Computing in Science & Engineering [see also IEEE Computational Science and Engineering] , Volume: 3 , Issue: 5 , Sept.-Oct. 2001  
 Pages:98 - 104

[\[Abstract\]](#)    [\[PDF Full-Text \(336 KB\)\]](#)    **IEEE JNL**


Print Format

[Home](#) | [Log-out](#) | [Journals](#) | [Conference Proceedings](#) | [Standards](#) | [Search by Author](#) | [Basic Search](#) | [Advanced Search](#) | [Join IEEE](#) | [Web Account](#) | [New this week](#) | [OPAC Linking Information](#) | [Your Feedback](#) | [Technical Support](#) | [Email Alerting](#) | [No Robots Please](#) | [Release Notes](#) | [IEEE Online Publications](#) | [Help](#) | [FAQ](#) | [Terms](#) | [Back to Top](#)

Copyright © 2004 IEEE — All rights reserved



US Patent &amp; Trademark Office

[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide



[Feedback](#) [Report a problem](#) [Satisfaction survey](#)
Terms used loop and native and performanceFound **23,313** of **140,980**

Sort results by


[Save results to a Binder](#)
[Try an Advanced Search](#)

Display results


[Search Tips](#)
[Try this search in The ACM Guide](#)
☐ Open results in a new window

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

Relevance scale ☐ ☐ ☐ ☐ ☐

### 1 [Performance measurement of dynamically compiled Java executions](#)

Tia Newhall, Barton P. Miller

June 1999 **Proceedings of the ACM 1999 conference on Java Grande**Full text available: [pdf\(1.54 MB\)](#)Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)**Keywords:** Java, dynamic compilation, performance profiling tool

### 2 [The structure and performance of interpreters](#)

Theodore H. Romer, Dennis Lee, Geoffrey M. Voelker, Alec Wolman, Wayne A. Wong, Jean-Loup Baer, Brian N. Bershad, Henry M. Levy

September 1996 **Proceedings of the seventh international conference on Architectural support for programming languages and operating systems**, Volume 31 , 30 Issue 9 , 5Full text available: [pdf\(1.17 MB\)](#)Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Interpreted languages have become increasingly popular due to demands for rapid program development, ease of use, portability, and safety. Beyond the general impression that they are "slow," however, little has been documented about the performance of interpreters as a class of applications. This paper examines interpreter performance by measuring and analyzing interpreters from both software and hardware perspectives. As examples, we measure the MIPS, Java, Perl, and Tcl interpreters running an ...

### 3 [Design, implementation, and performance measurement of a native-mode ATM transport layer \(extended version\)](#)

R. Ahuja, S. Keshav, H. Saran

August 1996 **IEEE/ACM Transactions on Networking (TON)**, Volume 4 Issue 4Full text available: [pdf\(1.66 MB\)](#)Additional Information: [full citation](#), [references](#), [index terms](#)**Keywords:** AAL 5, asynchronous transfer mode, native-mode ATM, personal computer, transport layer

#### 4 Increasing the portability and re-usability of protocol code

Bobby Krupczak, Kenneth L. Calvert, Mostafa H. Ammar

August 1997 **IEEE/ACM Transactions on Networking (TON)**, Volume 5 Issue 4

Full text available:  pdf(283.64 KB) Additional Information: [full citation](#), [references](#), [index terms](#)

**Keywords:** portability, protocol deployment, protocol implementation, protocol subsystem

#### 5 Techniques for obtaining high performance in Java programs

Iffat H. Kazi, Howard H. Chen, Berdenia Stanley, David J. Lilja

September 2000 **ACM Computing Surveys (CSUR)**, Volume 32 Issue 3

Full text available:  pdf(816.13 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


This survey describes research directions in techniques to improve the performance of programs written in the Java programming language. The standard technique for Java execution is interpretation, which provides for extensive portability of programs. A Java interpreter dynamically executes Java bytecodes, which comprise the instruction set of the Java Virtual Machine (JVM). Execution time performance of Java programs can be improved through compilation, possibly at the expense of portability ...

**Keywords:** Java, Java virtual machine, bytecode-to-source translators, direct compilers, dynamic compilation, interpreters, just-in-time compilers

#### 6 Obtaining sequential efficiency for concurrent object-oriented languages

John Plevyak, Xingbin Zhang, Andrew A. Chien

January 1995 **Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

Full text available:  pdf(1.09 MB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Concurrent object-oriented programming (COOP) languages focus the abstraction and encapsulation power of abstract data types on the problem of concurrency control. In particular, pure fine-grained concurrent object-oriented languages (as opposed to hybrid or data parallel) provides the programmer with a simple, uniform, and flexible model while exposing maximum concurrency. While such languages promise to greatly reduce the complexity of large-scale concurrent programming, the popularity of ...

#### 7 An architectural framework for migration from CISC to higher performance platforms

Gabriel M. Silberman, Kemal Ebcioglu

August 1992 **Proceedings of the 6th international conference on Supercomputing**

Full text available:  pdf(2.04 MB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

We describe a novel architectural framework that allows software applications written for a given Complex Instruction Set Computer (CISC) to migrate to a different, higher performance architecture, without a significant investment on the part of the application user or developer. The framework provides a hardware mechanism for seamless switching between two instruction sets, resulting in a machine that enhances application performance while keeping the same program behavior (from a user perspective ...)

#### 8 Dynamic native optimization of interpreters

Gregory T. Sullivan, Derek L. Bruening, Iris Baron, Timothy Garnett, Saman Amarasinghe



June 2003 **Proceedings of the 2003 workshop on Interpreters, Virtual Machines and Emulators**

Full text available:  [pdf\(150.25 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

For domain specific languages, "scripting languages", dynamic languages, and for virtual machine-based languages, the most straightforward implementation strategy is to write an interpreter. A simple interpreter consists of a loop that fetches the next bytecode, dispatches to the routine handling that bytecode, then loops. There are many ways to improve upon this simple mechanism, but as long as the execution of the program is driven by a representation of the program other than as a stream of n ...

9 Session 17: architecture: Sunder: a programmable hardware prefetch architecture for numerical loops

Tzi-cker Chiueh

November 1994 **Proceedings of the 1994 ACM/IEEE conference on Supercomputing**


Full text available:  [pdf\(922.38 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#)

*Beyond data caching, data prefetching is by far the most effective way to address the memory access bottleneck associated with high-performance processors. This is particularly true for scientific programs whose working sets cannot be easily fit into the on-chip data cache. This paper proposes a new data prefetching architecture called **Sunder**, which combines the flexibility and accurateness of software prefetching and the transparency and low-overhead of hardware prefetching. Th ...*

10 The trade-off between implicit and explicit data distribution in shared-memory programming paradigms

Dimitrios S. Nikolopoulos, Eduard Ayguadé, Theodore S. Papatheodorou, Constantine D. Polychronopoulos, Jesús Labarta

June 2001 **Proceedings of the 15th international conference on Supercomputing**

Full text available:  [pdf\(289.84 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

This paper explores previously established and novel methods for scaling the performance of OpenMP on NUMA architectures. The spectrum of methods under investigation includes OS-level automatic page placement algorithms, dynamic page migration and manual data distribution. The trade-off that these methods face lies between performance and programming effort. Automatic page placement algorithms are transparent to the programmer, but may compromise memory access locality. Dynamic page migration is ...

**Keywords:** OpenMP, data distribution, operating systems, page migration, performance evaluation, runtime systems

11 Register tiling in nonrectangular iteration spaces

Marta Jiménez, José M. Llabería, Agustín Fernández

July 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 4

Full text available:  [pdf\(2.21 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Loop tiling is a well-known loop transformation generally used to expose coarse-grain parallelism and to exploit data reuse at the cache level. Tiling can also be used to exploit data reuse at the register level and to improve a program's ILP. However, previous proposals in the literature (as well as commercial compilers) are only able to perform multidimensional tiling for the register level when the iteration space is rectangular. In this article we present a new general algorithm to perform m ...

**Keywords:** Data reuse, locality, loop optimization, loop tiling, register level

**12 Resource widening versus replication: limits and performance-cost trade-off**

David López, Josep Llosa, Mateo Valero, Eduard Ayguadé

July 1998 **Proceedings of the 12th international conference on Supercomputing**Full text available:  pdf(1.23 MB) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)**13 Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling**

D. S. Henty

November 2000 **Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)**Full text available:  pdf(197.99 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)  
 [Publisher Site](#)

The current trend in HPC hardware is towards clusters of shared-memory (SMP) compute nodes. For applications developers the major question is how best to program these SMP clusters. To address this we study an algorithm from Discrete Element Modeling, parallelized using both the message-passing and shared-memory models simultaneously ("hybrid" parallelization). The natural load-balancing methods are different in the two parallel models, the shared-memory method being in princip ...

**14 Improving Java performance using hardware translation**


Ramesh Radhakrishnan, Ravi Bhargava, Lizy K. John

June 2001 **Proceedings of the 15th international conference on Supercomputing**Full text available:  pdf(254.91 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

State of the art Java Virtual Machines with Just-In-Time (JIT) compilers make use of advanced compiler techniques, run-time profiling and adaptive compilation to improve performance. However, these techniques for alleviating performance bottlenecks are more effective in long running workloads, such as server applications. Short running Java programs, or client workloads, spend a large fraction of their execution time in compilation instead of useful execution when run using JIT compilers. In ...

**15 A comparison of empirical and model-driven optimization**

Kamen Yotov, Xiaoming Li, Gang Ren, Michael Cibulskis, Gerald DeJong, Maria Garzaran, David Padua, Keshav Pingali, Paul Stodghill, Peng Wu

May 2003 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation**, Volume 38 Issue 5Full text available:  pdf(448.74 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Empirical program optimizers estimate the values of key optimization parameters by generating different program versions and running them on the actual hardware to determine which values give the best performance. In contrast, conventional compilers use models of programs and machines to choose these parameters. It is widely believed that model-driven optimization does not compete with empirical optimization, but few quantitative comparisons have been done to date. To make such a comparison, we ...

**Keywords:** BLAS, blocking, code generation, compilers, empirical optimization, memory hierarchy, model-driven optimization, program transformation, tiling, unrolling

**16 Nonlinear array layouts for hierarchical memory systems**

Siddhartha Chatterjee, Vibhor V. Jain, Alvin R. Lebeck, Shyam Mundhra, Mithuna Thottethodi  
May 1999 **Proceedings of the 13th international conference on Supercomputing**

Full text available:  pdf(2.20 MB) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

**17 Speculative execution: A new speculation technique to optimize floating-point performance while preserving bit-by-bit reproducibility**

Mikio Takeuchi, Hideaki Komatsu, Toshio Nakatani

June 2003 **Proceedings of the 17th annual international conference on Supercomputing**

Full text available:  pdf(227.01 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

The bit-by-bit reproducibility of floating-point results, which is defined by the IEEE 754 standard, prohibits optimizations such as reassociation and the use of native operations such as fused multiply-add (FMA), and thus it significantly impairs floating-point performance. Recent network-oriented languages such as Java strictly conform to the standard, and thus their numerical computing performance becomes inherently lower than conventional languages. In this paper, we propose a new software te ...

**Keywords:** IA-64, IEEE 754, Java, accuracy, bit-by-bit reproducibility, floating-point speculation, fused multiply-add, instruction-level parallelism, just-in-time compiler, loop unrolling, prefetching, privatization, reassociation, software pipelining, striding

**18 AJaPACK: experiments in performance portable parallel Java numerical libraries**

Shigeo Itou, Satoshi Matsuoka, Hirokazu Hasegawa

June 2000 **Proceedings of the ACM 2000 conference on Java Grande**

Full text available:  pdf(976.22 KB) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

**19 Performance issues of scientific programming in Ada 95**

James B. White

November 1997 **Proceedings of the conference on TRI-Ada '97**

Full text available:  pdf(1.35 MB) Additional Information: [full citation](#), [references](#), [index terms](#)

**20 The NINJA project**

José E. Moreira, Samuel P. Midkiff, Manish Gupta, Pedro V. Artigas, Peng Wu, George Almasi

October 2001 **Communications of the ACM**, Volume 44 Issue 10

Full text available:  pdf(170.38 KB)  html(36.55 KB) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)


[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide



THE ACM DIGITAL LIBRARY


[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

 Terms used **caffeine** and **native** and **branch**

Found 1,476 of 140,980

Sort results by


[Save results to a Binder](#)

Display results


[Search Tips](#)
☐ Open results in a new window

[Try an Advanced Search](#)
[Try this search in The ACM Guide](#)


Results 1 - 20 of 200

 Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown


 Relevance scale ☐ ☐ ☐ ☐ ☐



- 1 [How java programs interact with virtual machines at the microarchitectural level](#)   
 Lieven Eeckhout, Andy Georges, Koen De Bosschere  
 October 2003 **ACM SIGPLAN Notices , Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications**, Volume 38 Issue 11

 Full text available:  [pdf\(348.88 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


Java workloads are becoming increasingly prominent on various platforms ranging from embedded systems, over general-purpose computers to high-end servers. Understanding the implications of all the aspects involved when running Java workloads, is thus extremely important during the design of a system that will run such workloads. In other words, understanding the interaction between the Java application, its input and the virtual machine it runs on, is key to a succesful design. The goal of this ...

**Keywords:** Java workloads, performance analysis, statistical data analysis, virtual machine technology, workload characterization

- 2 [Java bytecode to native code translation: the caffeine prototype and preliminary results](#)   
 Cheng-Hsueh A. Hsieh, John C. Gyllenhaal, Wen-mei W. Hwu  
 December 1996 **Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture**

 Full text available:  [pdf\(1.03 MB\)](#)  [Publisher Site](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

The Java bytecode language is emerging as a software distribution standard. With major vendors committed to porting the Java run-time environment to their platforms, programs in Java bytecode are expected to run without modification on multiple platforms. These first generation run-time environments rely on an interpreter to bridge the gap between the bytecode instructions and the native hardware. This interpreter approach is sufficient for specialized applications such as Internet browsers wher ...

- 3 [Exploiting Java instruction/thread level parallelism with horizontal multithreading](#)   
 Kenji Watanabe, Wanming Chu, Yamin Li  
 January 2001 **Australian Computer Science Communications , Proceedings of the 6th Australasian conference on Computer systems architecture**, Volume 23 Issue 4

 Full text available:  [pdf\(787.34 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#)

Java bytecodes can be executed with the following three methods: a Java interpreter running on a particular machine interprets bytecodes; a Just-In-Time (JIT) compiler translates bytecodes to the native primitives of the particular machine and the machine executes the translated codes; and a Java processor executes bytecodes directly. The first two methods require no special hardware support for the execution of Java bytecodes and are widely used currently. The last method requires an embedded J ...

#### 4 Techniques for obtaining high performance in Java programs

Iffat H. Kazi, Howard H. Chen, Berdenia Stanley, David J. Lilja  
September 2000 **ACM Computing Surveys (CSUR)**, Volume 32 Issue 3

Full text available:  [pdf\(816.13 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


This survey describes research directions in techniques to improve the performance of programs written in the Java programming language. The standard technique for Java execution is interpretation, which provides for extensive portability of programs. A Java interpreter dynamically executes Java bytecodes, which comprise the instruction set of the Java Virtual Machine (JVM). Execution time performance of Java programs can be improved through compilation, possibly at the expense of portability ...

**Keywords:** Java, Java virtual machine, bytecode-to-source translators, direct compilers, dynamic compilation, interpreters, just-in-time compilers

#### 5 Using complete system simulation to characterize SPECjvm98 benchmarks

Tao Li, Lizy Kurian John, Vijaykrishnan Narayanan, Anand Sivasubramaniam, Jyotsna Sabarinathan, Anupama Murthy

May 2000 **Proceedings of the 14th international conference on Supercomputing**

Full text available:  [pdf\(1.66 MB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Complete system simulation to understand the influence of architecture and operating systems on application execution has been identified to be crucial for systems design. While there have been previous attempts at understanding the architectural impact of Java programs, there has been no prior work investigating the operating system (kernel) activity during their executions. This problem is particularly interesting in the context of Java since it is not only the application that can invoke ...

#### 6 A single intermediate language that supports multiple implementations of exceptions

Norman Ramsey, Simon Peyton Jones

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation**, Volume 35 Issue 5

Full text available:  [pdf\(900.75 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

We present mechanisms that enable our compiler-target language, C--, to express four of the best known techniques for implementing exceptions, all within a single, uniform framework. We define the mechanisms precisely, using a formal operational semantics. We also show that exceptions need not require special treatment in the optimizer; by introducing extra dataflow edges, we make standard optimization techniques work even on programs that use exceptions. Our approach clarifies the design ...

#### 7 Technical correspondence: The simplest heuristics may be the best in Java JIT compilers


Jonathan L. Schilling

February 2003 **ACM SIGPLAN Notices**, Volume 38 Issue 2

Full text available:  pdf(1.00 MB) Additional Information: [full citation](#), [abstract](#), [references](#)

The simplest strategy in Java just-in-time (JIT) compilers is to compile each Java method the first time it is called. However, better performance can often be obtained by selectively compiling methods based on heuristics of how often they are likely to be called during the rest of the program's execution. Various heuristics are examined when used as part of the Caldera UNIX Java JIT compiler. The simplest heuristics involving the number of times the method has executed so far and the size of th ...

**Keywords:** JIT, Java, heuristics, just-in-time compiler, performance, selective compilation

- 8 [Design, implementation, and evaluation of optimizations in a just-in-time compiler](#)   
Kazuaki Ishizaki, Motohiro Kawahito, Toshiaki Yasue, Mikio Takeuchi, Takeshi Ogasawara,  
Toshio Suganuma, Tamiya Onodera, Hideaki Komatsu, Toshio Nakatani  
June 1999 **Proceedings of the ACM 1999 conference on Java Grande**

Full text available:  pdf(1.09 MB) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

- 9 [Compiling scheme to JVM bytecode:: a performance study](#) 


Bernard Paul Serpette, Manuel Serrano

September 2002 **ACM SIGPLAN Notices , Proceedings of the seventh ACM SIGPLAN international conference on Functional programming**, Volume 37 Issue 9

Full text available:  pdf(298.96 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


We have added a Java virtual machine (henceforth JVM) bytecode generator to the optimizing Scheme-to-C compiler Bigloo. We named this new compiler BiglooJVM. We have used this new compiler to evaluate how suitable the JVM bytecode is as a target for compiling strict functional languages such as Scheme. In this paper, we focus on the performance issue. We have measured the execution time of many Scheme programs when compiled to C and when compiled to JVM. We found that for each benchmark, at leas ...

**Keywords:** Java virtual machine, compilation, functional languages, scheme

- 10 [Optimizing indirect branch prediction accuracy in virtual machine interpreters](#) 

M. Anton Ertl, David Gregg

May 2003 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation**, Volume 38 Issue 5

Full text available:  pdf(190.05 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#), [review](#)

Interpreters designed for efficiency execute a huge number of indirect branches and can spend more than half of the execution time in indirect branch mispredictions. Branch target buffers are the best widely available form of indirect branch prediction; however, their prediction accuracy for existing interpreters is only 2%--50%. In this paper we investigate two methods for improving the prediction accuracy of BTBs for interpreters: replicating virtual machine (VM) instructions and combining seq ...

**Keywords:** branch prediction, branch target buffer, code replication, interpreter, superinstruction

- 11 [Dynamic native optimization of interpreters](#) 

Gregory T. Sullivan, Derek L. Bruening, Iris Baron, Timothy Garnett, Saman Amarasinghe

June 2003 **Proceedings of the 2003 workshop on Interpreters, Virtual Machines and**

## Emulators


Full text available:  [pdf\(150.25 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

For domain specific languages, "scripting languages", dynamic languages, and for virtual machine-based languages, the most straightforward implementation strategy is to write an interpreter. A simple interpreter consists of a loop that fetches the next bytecode, dispatches to the routine handling that bytecode, then loops. There are many ways to improve upon this simple mechanism, but as long as the execution of the program is driven by a representation of the program other than as a stream of n ...

### 12 Research sessions: path indexing: Covering indexes for branching path queries

Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, Henry F Korth

June 2002 **Proceedings of the 2002 ACM SIGMOD international conference on Management of data**


Full text available:  [pdf\(1.37 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

In this paper, we ask if the traditional relational query acceleration techniques of summary tables and covering indexes have analogs for branching path expression queries over tree- or graph-structured XML data. Our answer is yes --- the forward-and-backward index already proposed in the literature can be viewed as a structure analogous to a summary table or covering index. We also show that it is the smallest such index that covers all branching path expression queries. While this index is ver ...

### 13 An architectural framework for migration from CISC to higher performance platforms

Gabriel M. Silberman, Kemal Ebcioglu

August 1992 **Proceedings of the 6th international conference on Supercomputing**

Full text available:  [pdf\(2.04 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

We describe a novel architectural framework that allows software applications written for a given Complex Instruction Set Computer (CISC) to migrate to a different, higher performance architecture, without a significant investment on the part of the application user or developer. The framework provides a hardware mechanism for seamless switching between two instruction sets, resulting in a machine that enhances application performance while keeping the same program behavior (from a user per ...

### 14 An out-of-order execution technique for runtime binary translators

Bich C. Le

October 1998 **Proceedings of the eighth international conference on Architectural support for programming languages and operating systems**, Volume 32 , 33  
Issue 5 , 11

Full text available:  [pdf\(1.04 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

A dynamic translator emulates an instruction set architecture by translating source instructions to native code during execution. On statically-scheduled hardware, higher performance can potentially be achieved by reordering the translated instructions; however, this is a challenging transformation if the source architecture supports precise exception semantics, and the user-level program is allowed to register exception handlers. This paper presents a software technique which allows a translator ...

### 15 Dynamo: a transparent dynamic optimization system

Vasanth Bala, Evelyn Duesterwald, Sanjeev Banerjia

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation**, Volume 35 Issue 5

Full text available:  [pdf\(156.03 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index](#)

terms

We describe the design and implementation of Dynamo, a software dynamic optimization system that is capable of transparently improving the performance of a native instruction stream as it executes on the processor. The input native instruction stream to Dynamo can be dynamically generated (by a JIT for example), or it can come from the execution of a statically compiled native binary. This paper evaluates the Dynamo system in the latter, more challenging situation, in order to emphasize the ...

**16 Native code compilation of Erlang's bit syntax**

Per Gustafsson, Konstantinos Sagonas

October 2002 **Proceedings of the 2002 ACM SIGPLAN workshop on Erlang**


Full text available:  pdf(196.81 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#)

Erlang's bit syntax caters for flexible pattern matching on bit streams (objects known as *binaries*). Binaries are nowadays heavily used in typical Erlang applications such as protocol programming, which in turn has created a need for efficient support of the basic operations on binaries. To this effect, we describe a scheme for efficient native code compilation of Erlang's bit syntax. The scheme relies on *partial translation* for avoiding code explosion, an ...

**17 Efficient and language-independent mobile programs**

Ali-Reza Adl-Tabatabai, Geoff Langdale, Steven Lucco, Robert Wahbe

May 1996 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1996 conference on Programming language design and implementation**, Volume 31 Issue 5


Full text available:  pdf(1.03 MB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

This paper evaluates the design and implementation of Omniware: a safe, efficient, and language-independent system for executing mobile program modules. Previous approaches to implementing mobile code rely on either language semantics or abstract machine interpretation to enforce safety. In the former case, the mobile code system sacrifices universality to gain safety by dictating a particular source language or type system. In the latter case, the mobile code system sacrifices performance to ga ...

**18 Split-stream dictionary program compression**

Steven Lucco

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation**, Volume 35 Issue 5

Full text available:  pdf(89.99 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

This paper describes split-stream dictionary (SSD) compression, a new technique for transforming programs into a compact, interpretable form. We define a compressed program as interpretable when it can be decompressed at basic-block granularity with reasonable efficiency. The granularity requirement enables interpreters or just-in-time (JIT) translators to decompress basic blocks incrementally during program execution. Our previous approach to interpretable compression, the Byte-coded RISC ...

**Keywords:** compression, runtime system, virtual machine

**19 Measuring Experimental Error in Microprocessor Simulation**

Rajagopalan Desikan, Doug Burger, Stephen W. Keckler

June 2001 **Proceedings of the 28th annual international symposium on Computer architecture**

Full text available:  pdf(237.69 KB)



[Publisher Site](#)Additional Information: [full citation](#), [abstract](#), [citations](#), [index terms](#)

Abstract: We measure the experimental error that arises from the use of non-validated simulators in computer architecture research, with the goal of increasing the rigor of simulation-based studies. We describe the methodology that we used to validate a microprocessor simulator against a Compaq DS-10L workstation, which contains an Alpha 21264 processor. Our evaluation suite consists of a set of 21 microbenchmarks that stress different aspects of the 21264 microarchitecture. Using the microbenc ...

## 20 [Measuring experimental error in microprocessor simulation](#)

Rajagopalan Desikan, Doug Burger, Stephen W. Keckler

May 2001 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 2001 symposium on Software reusability: putting software reuse in context,**

Volume 26 Issue 3

Full text available: [pdf\(1.03 MB\)](#)

Additional Information: [full citation](#), [references](#), [index terms](#)



Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads: [Adobe Acrobat](#) [QuickTime](#) [Windows Media Player](#) [Real Player](#)



US Patent &amp; Trademark Office

[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide

THE ACM DIGITAL LIBRARY


[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

## Toba: Java For Applications: A Way Ahead of Time (WAT) Compiler

**Source**      [Technical Report: TR97-01](#)  
                  [Year of Publication: 1997](#)

**Authors**    [Todd A. Proebsting](#)  
                  [Gregg Townsend](#)  
                  [Patrick Bridges](#)  
                  [John H. Hartman](#)  
                  [Tim Newsham](#)  
                  [Scott A. Watterson](#)

**Publisher**   [University of Arizona](#)   [Tucson, AZ, USA](#)

**Additional Information:** [abstract](#)   [citations](#)   [collaborative colleagues](#)

**Tools and Actions:**    [Discussions](#)    [Find similar Technical Reports](#)    [Review this Technical Report](#)  
                                  [Save this Technical Report to a Binder](#)    [Display in BibTex Format](#)

### ↑ **ABSTRACT**

Toba is a system for generating efficient standalone Java applications. Toba includes a Java-bytecode-to-C compiler, a garbage collector, a threads package, and Java API support. Toba-compiled Java applications execute 1.5--10 times faster than interpreted and Just-In-Time compiled applications.

### ↑ **CITINGS 4**

[Bill Lin, Software synthesis of process-based concurrent programs, Proceedings of the 35th annual conference on Design automation conference, p.502-505, June 15-19, 1998, San Francisco, California, United States](#)

[Frederick Smith , Greg Morrisett, Comparing mostly-copying and mark-sweep conservative collection, ACM SIGPLAN Notices, v.34 n.3, p.68-78, March 1999](#)

[Matt Newsome , Des Watson, Proxy compilation of dynamically loaded Java classes with MoJo, ACM SIGPLAN Notices, v.37 n.7, July 2002](#)

[Ole Agesen , David Detlefs , Alex Garthwaite , Ross Knippel , Y. S. Ramakrishna , Derek White, An efficient meta-lock for implementing ubiquitous synchronization, ACM SIGPLAN Notices, v.34 n.10, p.207-222, Oct. 1999](#)

### ↑ **Collaborative Colleagues:**

<a href="#">Patrick Bridges:</a>	<a href="#">Andy Bavier</a>	<a href="#">Larry P. Peterson</a>
	<a href="#">Peter Bigot</a>	<a href="#">Rob Piltz</a>
	<a href="#">Peter A. Bigot</a>	<a href="#">Todd Proebsting</a>

	<a href="#"><u>John Hartman</u></a>	<a href="#"><u>Todd A. Proebsting</u></a>	
	<a href="#"><u>John H. Hartman</u></a>	<a href="#"><u>Oliver Spatscheck</u></a>	
	<a href="#"><u>John J. Hartman</u></a>	<a href="#"><u>Gregg Townsend</u></a>	
	<a href="#"><u>Brady Montz</u></a>	<a href="#"><u>Scott A. Watterson</u></a>	
	<a href="#"><u>Tim Newsham</u></a>		
	<a href="#"><u>Larry Peterson</u></a>		
	<a href="#"><u>Larry L. Peterson</u></a>		
<a href="#"><u>John H. Hartman:</u></a>	<a href="#"><u>Mary G. Baker</u></a>	<a href="#"><u>Randy H. Katz</u></a>	<a href="#"><u>Rob Piltz</u></a>
	<a href="#"><u>Scott Baker</u></a>	<a href="#"><u>Michael D. Kupfer</u></a>	<a href="#"><u>Todd Proebsting</u></a>
	<a href="#"><u>Andy Bavier</u></a>	<a href="#"><u>Edward K Lee</u></a>	<a href="#"><u>Todd A. Proebsting</u></a>
	<a href="#"><u>Peter Bigot</u></a>	<a href="#"><u>Ethan L. Miller</u></a>	<a href="#"><u>Prasenjit Sarkar</u></a>
	<a href="#"><u>Patrick Bridges</u></a>	<a href="#"><u>Brady Montz</u></a>	<a href="#"><u>Ken W. Shirriff</u></a>
	<a href="#"><u>Peter M Chen</u></a>	<a href="#"><u>Tim Newsham</u></a>	<a href="#"><u>Tammo Spalink</u></a>
	<a href="#"><u>Ann L. C Drapeau</u></a>	<a href="#"><u>John K. Ousterhout</u></a>	<a href="#"><u>Oliver Spatscheck</u></a>
	<a href="#"><u>Garth Gibson</u></a>	<a href="#"><u>David A. Patterson</u></a>	<a href="#"><u>Rajesh Sundaram</u></a>
	<a href="#"><u>Garth A. Gibson</u></a>	<a href="#"><u>Larry L. Peterson</u></a>	<a href="#"><u>Gregg Townsend</u></a>
	<a href="#"><u>Jørgen S. Hansen</u></a>	<a href="#"><u>Larry P. Peterson</u></a>	<a href="#"><u>Scott A. Watterson</u></a>
<a href="#"><u>Tim Newsham:</u></a>	<a href="#"><u>Patrick Bridges</u></a>		
	<a href="#"><u>John H. Hartman</u></a>		
	<a href="#"><u>Todd A. Proebsting</u></a>		
	<a href="#"><u>Gregg Townsend</u></a>		
	<a href="#"><u>Scott A. Watterson</u></a>		
<a href="#"><u>Todd A. Proebsting:</u></a>	<a href="#"><u>Andy Bavier</u></a>	<a href="#"><u>Charles N. Fischer</u></a>	<a href="#"><u>Larry L. Peterson</u></a>
	<a href="#"><u>Achyutram Bhamidipaty</u></a>	<a href="#"><u>Christopher W. Fraser</u></a>	<a href="#"><u>Rob Piltz</u></a>
	<a href="#"><u>Peter A. Bigot</u></a>	<a href="#"><u>David R. Hanson</u></a>	<a href="#"><u>Oliver Spatscheck</u></a>
	<a href="#"><u>Patrick Bridges</u></a>	<a href="#"><u>John H. Hartman</u></a>	<a href="#"><u>Rajesh Sundaram</u></a>
	<a href="#"><u>Christian S. Collberg</u></a>	<a href="#"><u>John J. Hartman</u></a>	<a href="#"><u>Gregg Townsend</u></a>
	<a href="#"><u>Sean Davey</u></a>	<a href="#"><u>Robert R. Henry</u></a>	<a href="#"><u>Gregg M. Townsend</u></a>
	<a href="#"><u>Saumya K. Debray</u></a>	<a href="#"><u>Steven M. Kurlander</u></a>	<a href="#"><u>Scott A. Watterson</u></a>
	<a href="#"><u>Dawson R. Engler</u></a>	<a href="#"><u>Steven Lucco</u></a>	<a href="#"><u>Benjamin R. Whaley</u></a>
	<a href="#"><u>Jens Ernst</u></a>	<a href="#"><u>Brady Montz</u></a>	
	<a href="#"><u>William Evans</u></a>	<a href="#"><u>Tim Newsham</u></a>	
<a href="#"><u>Gregg Townsend:</u></a>	<a href="#"><u>Gregory R. Andrews</u></a>	<a href="#"><u>Todd A. Proebsting</u></a>	
	<a href="#"><u>Patrick Bridges</u></a>	<a href="#"><u>Titus Purdin</u></a>	
	<a href="#"><u>Michael Coffin</u></a>	<a href="#"><u>Scott A. Watterson</u></a>	
	<a href="#"><u>Irving Elshoff</u></a>		
	<a href="#"><u>William Evans</u></a>		
	<a href="#"><u>John H. Hartman</u></a>		
	<a href="#"><u>David Kirkpatrick</u></a>		
	<a href="#"><u>Tim Newsham</u></a>		
	<a href="#"><u>Kelvin Nilson</u></a>		
	<a href="#"><u>Ronald A. Olsson</u></a>		
<a href="#"><u>Scott A. Watterson:</u></a>	<a href="#"><u>Patrick Bridges</u></a>		
	<a href="#"><u>Saumya K. Debray</u></a>		
	<a href="#"><u>John H. Hartman</u></a>		
	<a href="#"><u>Robert Muth</u></a>		
	<a href="#"><u>Tim Newsham</u></a>		
	<a href="#"><u>Todd A. Proebsting</u></a>		
	<a href="#"><u>Gregg Townsend</u></a>		

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)